A METHOD OF, AND SYSTEM FOR, HEURISTICALLY DETECTING VIRUSES IN EXECUTABLE CODE

The present invention relates to a method of, and system for, heuristically detecting viruses in executable code by detecting that an executable file is likely to be a previous known executable file, and that the file has been changed. This technique is especially applicable to situations where files enter a system, are checked, then leave, such as email gateways or web proxies. However, it is not intended to be limited to those situations.

The expression "virus" as used in this specification and claims is to be understood in a broad, inclusive sense encompassing any form of malware in executable code.

Increasing use of the Internet, personal computers and local- and wide-area networks has made the problem of computer viruses ever more acute.

Some internet service providers (ISP) offer anti-virus scanning, of attachments to e-mails, and file-downloads and transfers, as a value-added service to their clients. A conventional method of anti-virus scanning is to scan the file looking for patterns or sequences of bytes which have been established as being a characteristic "signature" of a known virus. However, signature-based scanning is not ideal in the rapidly changing internet environment particularly if it is used as the sole virus detection method. When an outbreak of a previously-unknown virus occurs, anti-virus specialists have first to identify a suitable signature to characterise the virus and this then has to be disseminated to anti-virus scanners in the field, all of which takes time. Another disadvantage is that file-scanning can very resource-intensive, particularly where file traffic volumes are high. The system needs to have enough processing power and file-buffering capacity to keep delays to a minimum and to cope with peak demands.

According to the present invention, there is provided an anti-virus file scanning system for computer files comprising:

a computer database containing records of known executable programs which are deemed to be uninfected and criteria by which a file being processed can be determined to be an instance of one of those programs;

means for processing a file being transferred between computers to determine whether the file matches the criteria characterising a file as an unchanged instance of a program in the database; and

means for signalling the file as known or not depending on the determination made by the processing means.

The invention also provides a method of anti-virus scanning system computer files comprising:

maintaining a computer database containing records of known executable programs which are deemed to be uninfected and criteria by which a file being processed can be determined to be an instance of one of those programs;

processing a file being transferred between computers to determine whether the file matches the criteria characterising a file as an unchanged instance of a program in the database; and

signalling the file as known or not depending on the determination made by the processing means.

The invention is based upon the fact that a significant proportion of network traffic of executable files is made up of uninfected copies of common applications and utilities such as WinZip and the like. If these can be reliably identified as such, the system need not scan them further, so reducing the processing and storage load on the system.

The invention will be further described by way of non-limitative example with reference to the accompanying drawings, in which

Figure 1 is a block diagram of the present invention.

Figure 1 illustrates one form of a system according to the present invention, which might be used, for example by an ISP as part of a larger anti-virus scanning system which employs additional scanning methods on files which are not filtered out as "safe" by the system of Figure 1.

The source of the files inputted to the system of Figure 1 is not material to the invention. The files could, for example, be copies of attachments of e-mails being processed by an ISP en route to delivery to its customers, which have been temporarily saved to disk for anti-virus processing before the e-mail is delivered. In the case of processing file downloads or transfers, the system could be associated with a trusted proxy server which retrieves the file from an untrusted remote site, e.g. a website or FTP site, saves it temporarily to disk for scanning and then delivers it to the user via whatever internet protocol he or she is using.

The nature of a file-infecting virus is such that it changes the contents of an executable file by inserting the virus code. On a file system, it is possible to detect this change by creating a checksum or hash string of the file (such as those generated by the

well-known MD5 or SHA5 checksum algorithms), and comparing it to the checksum that the file should have. If they are the same, then to an extremely high probability, the file is unchanged. If they are different, then the file has been changed.

However, this check is not possible if it is being carried out at a gateway, such as an email gateway. When the file arrives, the checksum of the file in its good state is not known – only the current checksum can be calculated. Therefore, there is no immediate method of determining that the file has been changed.

One way to get round this is to find a way of recognising the file. Once this is done, the checksum of the file in its good state can be looked up in a database. This known good checksum can then be compared with the actual checksum of the file. If they are different, the file has been changed, and special action may then be taken – this could for instance include quarantining the file, or marking it for further automatic or manual analysis. This is the method implemented by the system 100 of Figure 1, which operates according to the following algorithm.

1. A file arrives for scanning, perhaps as an email attachment, or a web download at an input 101.

2. The file is passed to an 'EXE recogniser' 102 to check if it is an executable file. If not, it is not analysed further, and is treated as an unknown file.

3. The file is then passed to a 'File recogniser' 103 to check if it is a previously known file by reference to a database 104 of known files. If not, it is not analysed further, and is signalled as an unknown file at an output 107.

4. The file is then passed to a 'Difference checker' 105 to check if it is an unchanged copy of the known file. If so, it is not analysed further, and is signalled as a known file at an output 108.

5. The file is passed to an 'Exception checker' 106 to check for known exceptions. If an exception list match is found, no further action is taken, and the file is signalled as an unknown file at the output 107.

6. The file appears to be a changed copy of a previously known executable file. It is therefore signalled as an example of file infecting malware at an output 109.

The system 101 may form part of a larger malware detection system which submits files initially to the system 101 and then, depending on the results signalled at outputs 107, 108 and 109 may submit the file to other file-scanning sub-systems if the file is signalled by output 107 to be an unknown file, treat the file as known and safe if

signalled as such by the output 108 or handle the file as malware if signalled as such at output 109. In the latter case, the file may be subject to any of the usual malware-handling actions (or any appropriate combination of them), e.g. it may be quarantined, the intended recipient may be notified that malware has been detected, it may be submitted for attention by a human operator, it may be deleted, etc.

In the case of a file signalled as "unknown" at output 107, once this has been processed further to determine whether it is in fact malware, if the determination is that it is not, the database 104 may be automatically updated with an entry for it, so that the system 101 can treat subsequent instances of it as a known file.

One way of implementing the EXE recogniser 102 is simply by comparing the first few bytes of the file with the published specifications for the particular format.

For instance, here is a simplistic example of an algorithm for determining if a file is likely to be a Windows PE file.

Read in first 2 bytes. If these are not 'MZ' then stop
Read in another 58 bytes.
Read in 4 bytes into variable x (treating using intel byte-ordering)
Seek to offset x in file
Read in 4 bytes
If bytes are P E \0 \0, then file is likely to be a Windows PE file

Recognisers can be created for each additional executable file format as desired.

The file recogniser 1 can use various strategies for determining if the file is a known file. For instance, pattern matching scanning techniques can be used. This is similar to the techniques used by virus scanners to identify known malware – except in this case to identify known good files instead. Another technique is to split the file into areas suggested by analysing the structure of the file, and checksumming each of the areas. These can then be compared with checksums in the database 104 which have been generated by doing the same analysis on known files. Unless the virus modifies every single area, there will be at least one match which will then identify the file.

A simplistic implementation of the difference checker 105 could checksum the entire file, and compare this with the checksum the known file should have.

It is possible that occasional false positives could arise. For instance, although unlikely, two different files could have the same checksum. Another scenario is a packer compressing part of a file, but leaving others uncompressed. The exception checker

106 can be designed to accommodate these situations. For instance, in the case of the packed files, all files packed by the particular packer exhibiting the behaviour can be ignored. It will be apparent from Figure 1 that this treatment of packers does not constitute a "hole" in the system from the point of view of letting viruses through. The file is only processed by the exception checker if it has already been determined that the file does not correspond to an unchanged version of a known file, and what the exception checker does is to signal whether the file a) should be considered as malware, if it is not in the exception list or b) is unknown and should therefore be subject to further virus-detection processing.

As well as using the system 101 as a stand-alone virus detection algorithm, it can be combined with systems implementing other techniques as part of a larger system. For instance, programs flagged as malware by the system 101 may be allocated a certain score, or variety of scores depending which tests pass and fail. Scores may also be assigned using other heuristic techniques, and only if the total score passes some limit is the program flagged as viral.

The system 101 can also be used as an indicator for program files which may need further analysis. For instance, programs flagged as known files at an output 108 may not need further analysis, since they have already been flagged as 'safe'. Programs flagged at output 107 as unknown files may need further analysis.

The difference checker 105 can also be adapted to cope with files that are expected to change. For instance, a self extracting ZIP file may have a potentially different ZIP bound to it every time. Some programs also carry within themselves registration keys, and these may be different for every user. In such cases it is possible to create a checksum of the parts of the file that are invariant each time, and use that. For instance, the checksum may be created from bytes 0x0000 to 0x0A00, and from 0x0A73 to 0x3000.

If the difference checker 105 is modified in this way, then the files it matches will not necessarily be treated as known files needing no further analysis. For instance, if a self extracting ZIP file is recognised, then the files inside the ZIP archive will need extracting and analysing before the file can be considered safe. Each known file will therefore need an associated status which flags whether further processing is or is not required.